Optimal Projection Method in Sphere Decoding

Arash Ghasemmehdi and Erik Agrell

***Abstract*—An entirely different approach to complexity reduction in sphere decoders is taken. Here we demonstrate that most of the calculations in the standard algorithms are in fact redundant in the sense that the calculated values are never used. This applies to all recursive sphere decoder algorithms, including the numerous variations of the Fincke-Pohst and Schnorr-Euchner strategies. We propose a method, which is applicable to lattices as well as finite constellations, to avoid these redundant calculations, thus reducing the complexity. We emphasize that the algorithms otherwise perform exactly as before, visiting the same points in the same order, and returning the same result. Pseudocode is given to facilitate immediate implementation. In simulation results, it is shown that the relative complexity gain with the proposed add-on goes up linearly as the dimension of the lattice increases. For instance, the complexity is reduced to one fourth for lattices at dimension sixty.**

***Index Terms*—Closest point search, Fincke-Pohst, lattice, Lenstra-Lenstra-Lovász (LLL) reduction, maximum likelihood (ML) detection, multiple-input multiple-output (MIMO), Schnorr-Euchner, sphere decoder, Voronoi region.**

I. Introduction

EVERY lattice is represented with its generator matrix $\boldsymbol{G}$, whose entries are real numbers. Let n and m denote the number of rows and columns of $\boldsymbol{G}$ respectively with $n \leq m$. The rows of $\boldsymbol{G}$, which are $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, are called basis vectors and are assumed to be linearly independent vectors in $\mathbb{R}^m$. The lattice of dimension n is defined as the set of points

$$\Lambda(\boldsymbol{G}, \mathbb{Z}) = \{u_1\boldsymbol{b}_1 + \ldots + u_n\boldsymbol{b}_n \mid u_i \in \mathbb{Z}\}. \tag{1}$$

Every point is surrounded by a region, which is known as the Voronoi region. The Voronoi region of a lattice point is the set of all vectors in $\mathbb{R}^m$ that has a shorter Euclidean distance to this lattice point than to any other point in $\Lambda(\boldsymbol{G}, \mathbb{Z})$. Hence, the set of all Voronoi regions tile the space $\mathbb{R}^m$ without overlap, disregarding the boundaries.

Finding the closest lattice point to a given vector $\boldsymbol{r} \in \mathbb{R}^m$ is equivalent to finding the Voronoi region that this vector belongs to. It requires minimization of the metric $\|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$ over all lattice points $\boldsymbol{u}\boldsymbol{G}$. However, complete enumeration of the points is not feasible. The main idea of closest point search algorithms is to minimize the metric over all lattice points located inside a hypersphere centered on $\boldsymbol{r}$, and reduce the number of points that has to be enumerated. For lattices with known structural properties, closest point search algorithms could be modified to avoid some unnecessary numerical operations [1], [2]. However, in our case where lattices without any specific structures are addressed we have to implement a brute force search to find the nearest lattice point [3], [4].

In 1981 Pohst [5] suggested a way of finding the closest point in lattices, which later on was complemented by Fincke and Pohst in [6]. The implementation details of Fincke-Pohst (FP) enumeration method were first presented by Viterbo and Biglieri in [7]. In 1999, Viterbo and Boutros applied the FP enumeration method to maximum likelihood (ML) detection for finite constellations [4]. Later on, Agrell et al. in [3] illustrated that the Schnorr-Euchner (SE) refinement [8] of the FP enumeration strategy improves the complexity of the sphere decoder algorithm. It combines the advantages of the Babai nearest plane algorithm [1] and the FP strategy.

A given lattice can be represented by many different sets of basis vectors. The efficiency of the closest point search algorithms can be enhanced if the basis vectors of the lattice are reduced. Reduction in lattices is a way of making the basis vectors as short as possible and fairly orthogonal to each other, without changing the structure of the lattice, in order to shorten the overall search time. In 1982 Lenstra, Lenstra, and Lovász [9] achieved a breakthrough by constructing an algorithm that very fast produces a reduced basis in a certain sense, a so-called LLL reduced basis, for any given lattice basis. The implementation details of LLL reduced basis is well presented in [9], [10].

During the last decade a lot of work has been done to improve the efficiency of the sphere decoder algorithms [11]–[15], due to the vast amount of demand they have in numerous types of applications. In communication theory, there are lots of applications where the closest point problem arises. ML detection in multiple-input multiple-output (MIMO) channels [11], [16]–[19], quantization [20], vector perturbation in multiuser communications [21], joint detection in direct-sequence multiple access system [22], multiple symbol differential detection [23], and Max-Log-Map detection [24] are some examples of them.

The closest point search algorithms can be modified to find the ML point in finite constellations [4], [11], which has an important application in MIMO channels. Assuming a system with n transmit and m receive antennas, the new set of points $\Lambda(\boldsymbol{G}, \mathcal{U})$ is defined by replacing $\mathbb{Z}$ in (1) with the finite range of integers

$$\mathcal{U} = \{U_{min}, U_{min} + 1, \ldots, U_{max}\}. \tag{2}$$

The transmit set can be mapped to an L-PAM constellation with $L = U_{max} - U_{min} + 1$. The received vector after an additive white Gaussian noise (AWGN) channel with double-sided noise power spectral density $N_0/2$ is

$$\boldsymbol{r} = \boldsymbol{u}\boldsymbol{G} + \boldsymbol{n}, \tag{3}$$

where $\boldsymbol{u} \in \mathcal{U}^n$, $\boldsymbol{r} \in \mathbb{R}^m$, $\boldsymbol{G} \in \mathbb{R}^{n \times m}$, and $\boldsymbol{n} \in \mathbb{R}^m$ is a vector of independent and identically distributed (i.i.d) Gaussian noise with variance $N_0/2$. In this case ML detection is equivalent to minimization of the metric $\|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$ over all possible points $\boldsymbol{u}\boldsymbol{G}$ with $\boldsymbol{u} \in \mathcal{U}^n$. In MIMO systems where usually quadrature amplitude modulation (QAM) is used, the L^2-QAM signal constellation can be viewed as two real-valued

L-PAM constellations with $\boldsymbol{u} \in \mathcal{U}^{2n}$, $\boldsymbol{r} \in \mathbb{R}^{2m}$, $\boldsymbol{G} \in \mathbb{R}^{2n\times 2m}$, and $\boldsymbol{n} \in \mathbb{R}^{2m}$.

In this paper we take a completely new approach to reduce the complexity of the sphere decoder algorithms, thus drawing attention to a hitherto unnoticed problem with the standard algorithms. It is illustrated that the proposed sphere decoder algorithms based on FP [6] and SE [8] enumeration strategies are not well optimized in the sense they perform many excessive numerical operations and calculate variables which are not supposed to be used. Thus, a method is proposed to prevent these unnecessary computations. However, the revision proposed is not related to choosing a more accurate upper bound, or scanning the layers optimally. We believe that the SE strategy is the best way in this regard. Our modification instead changes how the received vector $\boldsymbol{r}$ is projected onto the basis vectors, which accounts for most of the floating point calculations in the sphere decoder. With the method proposed, not a single value would be calculated twice or remain without any use. This is what we mean with an *optimal* projection method.

Based on the proposed modification an algorithm inspired by the SE enumeration strategy is established. A standalone representation of the algorithm is included in Sec. IV. The section also contains the modification needed to implement the renewed algorithm for finite constellations.

II. Closest Point Search Algorithms

A thorough conceptual description of the sphere decoder algorithms is presented here. Even though what is explained herein has the same basic principles as the previous enumeration methods explained in [3]–[8], for better understanding and comprehension of the main contribution of the paper, it seems indispensable.

Using the so-called $QR\ decomposition$, any real-valued $n \times m$ matrix $\boldsymbol{G}$ with $n \leq m$ can be factorized as $\boldsymbol{G} = \boldsymbol{RQ}$, where $\boldsymbol{R}$ is an $n \times n$ lower-triangular matrix and $\boldsymbol{QQ}^T = \boldsymbol{I}$. Using the matrix $\boldsymbol{R}$, which can be seen as a rotated and reflected version of $\boldsymbol{G}$, is much more convenient than using lattice generator matrix $\boldsymbol{G}$ itself. The obtained lattice point can then easily be shifted to its original place through rotation and reflection [3]. Hence, without loss of generality, we assume that $\boldsymbol{G}$ is a square lower-triangular matrix. For better explanation and visualization, the $\boldsymbol{H} = \boldsymbol{G}^{-1}$ matrix instead of $\boldsymbol{G}$ is used. The elements of $\boldsymbol{H}$ are named according to the convention

$$\boldsymbol{H} = \begin{bmatrix} H_{1,1} & 0 & \cdots & 0 \\ H_{2,1} & H_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ H_{n,1} & H_{n,2} & \cdots & H_{n,n} \end{bmatrix}, \tag{4}$$

where $\boldsymbol{H}$ is an $n \times n$ lower-triangular matrix with positive diagonal elements.

Every lattice can be divided into layers of lower-dimensional lattices. The diagonal elements of $\boldsymbol{H}$ illustrate the distances between these layers, such that $1/H_{i,i}$ represents the distance between the $(i-1)$-dimensional layers in an i-dimensional layer. Thus $1/H_{1,1}$ is the distance between the lattice points in a one-dimensional layer.

We assume that we have a received vector $\boldsymbol{r}$ in an n-dimensional Euclidean space $\mathbb{R}^n$ and that an upper bound C on $\|\boldsymbol{r} - \boldsymbol{uG}\|^2$ is somehow known. We intend to find the closest lattice point to this vector. As is obvious from the name sphere, the sphere decoder algorithms start by drawing a virtual n-dimensional hypersphere centered on $\boldsymbol{r}$ with radius $\sqrt{C}$ and then enumerating the lattice points located inside this hypersphere. By finding a new potential closest point, the radius of the hypersphere in which the points are enumerated is reduced, and the algorithm considers minimizing the metric over the lattice points inside the smaller hypersphere, also centered on $\boldsymbol{r}$, with improved radius. However, the way that the first radius is calculated differs between the FP and SE algorithms. For the SE enumeration strategy, the radius of the examined hypersphere is first considered as infinity. We then continue by finding the first potential closest point, which is the Babai point [25] in our case, and defining a new upper bound. This speeds up the algorithm compared to the FP enumeration method where defining the initial upper bound is a critical issue.

Fig. 1 illustrates an n-dimensional hypersphere with radius $\sqrt{C}$ centered on a received vector $\boldsymbol{r}$ which contains several $(n-1)$-dimensional layers. The basis vector $\boldsymbol{b}_n$ is in the same direction as the hypotenuse of right triangles $\triangle ABC$ and $\triangle DEC$, while all the other basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$ lie in the subspace spanned by one of these $(n-1)$-dimensional layers.

Starting from dimension n, after defining the initial upper bound, the received vector

$$\boldsymbol{r} = (r_1, r_2, \ldots, r_n) \in \mathbb{R}^n \tag{5}$$

is projected onto the lattice basis vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$. This can be easily done by a simple matrix multiplication

$$\boldsymbol{e}_n\boldsymbol{G} = \boldsymbol{r} \Rightarrow \boldsymbol{e}_n = \boldsymbol{rH}, \tag{6}$$

where $\boldsymbol{e}_n = (E_{n,1}, E_{n,2}, \ldots, E_{n,n}) \in \mathbb{R}^n$. Observe in particular that $E_{n,n} = r_n H_{n,n}$ because of the lower-triangular form (4). By knowing C and $E_{n,n}$ according to [4] the corresponding range for the integer component u_n, which is also intuitively conspicuous from Fig. 1, is

$$\lceil -H_{n,n}\sqrt{C} + E_{n,n} \rceil \leq u_n \leq \lfloor H_{n,n}\sqrt{C} + E_{n,n} \rfloor, \tag{7}$$

where $\lceil\ \rceil$ and $\lfloor\ \rfloor$ denote the round up and round down operations respectively. Another difference between the FP and SE enumeration methods shows up here. While the FP examines all the layers between the intervals above sequentially, the SE refinement is to follow the layers in a zigzag path. In other words, the SE algorithm first examines the nearest $(n-1)$-dimensional layer and then goes for the second nearest $(n-1)$-dimensional layer which is on the opposite side of $\boldsymbol{r}$, viewed from the nearest layer. This is well known as the main contribution of the SE refinement to the Pohst enumeration strategy, and it is the reason why the first lattice point visited by SE algorithm is the same regardless of C, whereas with the FP algorithm, the first point typically lies close to the boundary.

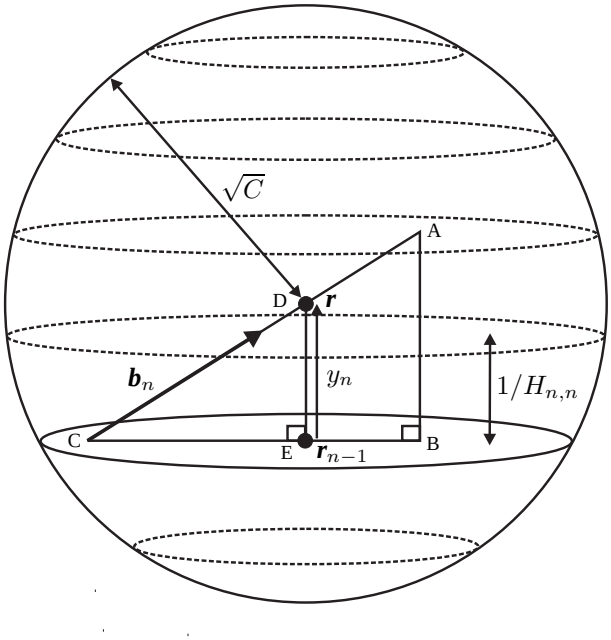

Fig. 1. Snapshot of an n-dimensional hypersphere, divided into a stack of $(n-1)$-dimensional hyperspheres (layers).

After finding the potential $(n-1)$-dimensional layer that is to be examined, the next step is to calculate the orthogonal displacement y_n from the received vector $\boldsymbol{r}$ to this layer, which is shown with line $\overline{DE}$ in Fig. 1. This can be performed by the similarity theorem in geometry:

$$\begin{aligned} &\triangle ABC \sim \triangle DEC \\ \Rightarrow\ & y_n = \frac{E_{n,n} - u_n}{H_{n,n}}. \end{aligned} \tag{8}$$

In order to calculate the $E_{n-1,n-1}$ value that will be used later on to calculate the u_{n-1} range (12) and y_{n-1} displacement (16), the received vector $\boldsymbol{r}$ is first projected to the examined $(n-1)$ dimensional layer (9) and then to the lattice basis vectors (10). We use the notation $\boldsymbol{r}_{n-1}$ for the projected received vector $\boldsymbol{r}$, where $n-1$ denotes the dimension of the layer that the received vector is projected on. Thanks to the lower-triangular representation, the orthogonal projection of $\boldsymbol{r}$ onto the $(n-1)$-dimensional layer currently being investigated affects only the last component of $\boldsymbol{r}$. So, it would be sufficient to subtract y_n from the nth element of $\boldsymbol{r}$ to reach

$$\boldsymbol{r}_{n-1} = (r_1, r_2, \ldots, r_n - y_n), \tag{9}$$

where according to (6) and (8)

$$r_n - y_n = \frac{u_n}{H_{n,n}}.$$

This positions $\boldsymbol{r}_{n-1}$ exactly on the perpendicular vertex of $\triangle DEC$. Projecting vector $\boldsymbol{r}_{n-1}$ to the lattice basis vectors can also be done by a simple matrix multiplication similar to (6)

$$\begin{aligned} \boldsymbol{e}_{n-1} &= \boldsymbol{r}_{n-1}\boldsymbol{H} && (10)\\ &= \boldsymbol{r}\boldsymbol{H} - (0, \ldots, 0, y_n)\boldsymbol{H} \\ &= \boldsymbol{e}_n - y_n(H_{n,1}, \ldots, H_{n,n}), && (11) \end{aligned}$$

where $\boldsymbol{e}_{n-1} = (E_{n-1,1}, \ldots, E_{n-1,n-1}, E_{n-1,n})$, and $E_{n-1,n} = E_{n,n} - y_n H_{n,n}$ which is also equal to u_n. Calculating $E_{n-1,n-1} = E_{n,n-1} - y_n H_{n,n-1}$, which is the value that should be multiplied to the lattice basis vector $\boldsymbol{b}_{n-1}$ to create the projected vector $\boldsymbol{r}_{n-1}$, and substituting $\lambda_n = y_n^2$ the corresponding range for u_{n-1} is [4]

$$\begin{aligned} \lceil -H_{n-1,n-1}\sqrt{B_{n-1}} + E_{n-1,n-1} \rceil &\le u_{n-1} \\ &\le \lfloor H_{n-1,n-1}\sqrt{B_{n-1}} + E_{n-1,n-1} \rfloor, \end{aligned} \tag{12}$$

where $B_{n-1} = C - \lambda_n$ is the squared radius of the examined $(n-1)$-dimensional layer.

The sphere decoder is applied recursively to search this $(n-1)$-dimensional layer. Thereafter the next u_n value in (7) is generated and a new $(n-1)$-dimensional layer is searched. This search strategy can be illustrated as a depth-first search [26] of the tree in Fig. 2. Once we reach the last node the bounds are updated recursively and the search backtracks to the most recent node that has not finished exploring yet.

By doing a simple generalization to compute the projection values in an i-dimensional layer, which are used later on in (16) and (20), for $i = 0, \ldots, n-1$ we derive

$$\begin{aligned} \boldsymbol{e}_i &= \boldsymbol{r}_i\boldsymbol{H} && (13)\\ &= \boldsymbol{e}_n - \sum_{j=i+1}^{n} y_j(H_{j,1}, \ldots, H_{j,n}) \\ &= \boldsymbol{e}_{i+1} - y_{i+1}(H_{i+1,1}, \ldots, H_{i+1,n}), && (14) \end{aligned}$$

where $\boldsymbol{r}_i$ is a projected received vector $\boldsymbol{r}$ to an i-dimensional layer, and

$$\boldsymbol{e}_i = (E_{i,1}, \ldots, E_{i,i}, u_{i+1}, \ldots, u_n) \tag{15}$$

gives the coefficients of $\boldsymbol{r}_i$ expressed as a linear combination of the lattice basis vectors. Thus in a zero-dimensional layer, which is a lattice point, $\boldsymbol{r}_0 \in \Lambda(\boldsymbol{G}, \mathbb{Z})$ and $\boldsymbol{e}_0 = \boldsymbol{r}_0\boldsymbol{H} \in \mathbb{Z}^n$. Assuming an i-dimensional sphere similar to Fig. 1, the orthogonal displacement between the projected vector $\boldsymbol{r}_i$ and the examined $(i-1)$-dimensional layer is

$$y_i = \frac{E_{i,i} - u_i}{H_{i,i}}, \qquad i = 1, \ldots, n. \tag{16}$$

Based on lower-triangular form (4) and the interpretation that y_i only affects the ith component of $\boldsymbol{r}_i$, for $i = 1, \ldots, n$

$$\boldsymbol{r}_{i-1} = (r_1, \ldots, r_{i-1}, r_i - y_i, \ldots, r_n - y_n), \tag{17}$$

where, according to (13) and (16)

$$r_i - y_i = \frac{u_i}{H_{i,i}} - \sum_{j=i+1}^{n} \frac{(r_j - y_j)H_{j,i}}{H_{i,i}}, \qquad i = 1, \ldots, n.$$

Similarly, the bounds for every i-dimensional layer are

$$\lambda_i = y_i^2 + y_{i+1}^2 + \ldots + y_n^2, \qquad i = 1, \ldots, n, \tag{18}$$

$$B_i = C - \lambda_{i+1}, \qquad i = 1, \ldots, n-1, \tag{19}$$

where λ_i is the squared distance from the received vector $\boldsymbol{r}$ to the projected vector $\boldsymbol{r}_{i-1}$, and B_i is the squared radius of the examined i-dimensional layer. Hence, it can be interpreted that λ_1 denotes the Euclidean distance between the received

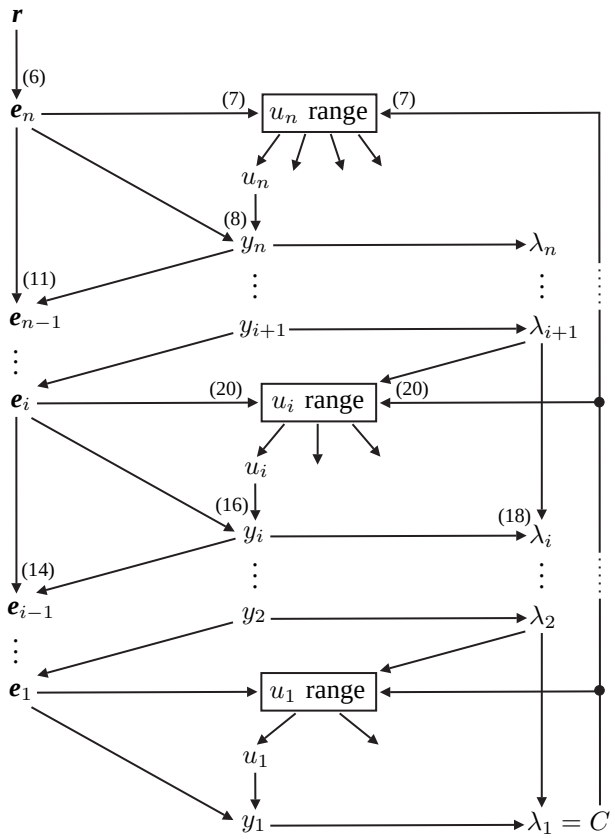

Fig. 2. The search tree of the sphere decoder, where arrows illustrate the connection between the different parameters. Only parameters computed in an implementation are included. Only one node at each level $1, \ldots, n$ is expanded.

vector $\boldsymbol{r}$ and a potential closest point $\boldsymbol{r}_0$. Finally, the ranges for the ith integer component of the u values for $i = 1, \ldots, n-1$ are [4]

$$\lceil -H_{i,i}\sqrt{B_i} + E_{i,i} \rceil \leq u_i \leq \lfloor H_{i,i}\sqrt{B_i} + E_{i,i} \rfloor, \qquad (20)$$

where $E_{i,i}$ is the value that should be multiplied to the lattice basis vector $\boldsymbol{b}_i$ to create the projected vector $\boldsymbol{r}_i$ (13).

III. Projection Of The Received Vector

In this section we propose our own modification to the SE closest point search algorithm, and also give a suggestion to improve the FP method in [4]. Even though the SE refinement to the FP strategy improved the complexity of the sphere decoder algorithm, there is still an area for improvement.

A. The Standard Projection Method

Most of the numerical operations carried out in standard sphere decoders are related to the projection of the received vector $\boldsymbol{r}$, or its lower-dimensional counterpart, onto the lattice basis vectors while we are continuously moving up and down in the hierarchy of layers and updating the $\boldsymbol{e}_i$ vectors in Fig. 2. The $E_{i,j}$ values of $\boldsymbol{e}$ vectors can be calculated and presented as

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{e}_n \end{bmatrix} = \begin{bmatrix} E_{1,1} & \cdots & E_{1,n} \\ \vdots & \ddots & \vdots \\ E_{n,1} & \cdots & E_{n,n} \end{bmatrix}.$$

All elements are updated from the elements immediately below, as shown in (14) and Fig. 2. However, the only values that are required in the sphere decoder algorithms are the diagonal elements $E_{i,i}$, used in (16) and (20). Thus, the elements located above the diagonal of $\boldsymbol{E}$ are not required to be calculated. They are corresponding elements to u values that have already been calculated in the previous stages of the algorithm, see (15). It is also obvious that $\boldsymbol{e}_n$ (6) is just calculated once since there exists just a single n-dimensional sphere.

The sphere decoder algorithm based on the SE enumeration strategy proposed in [3] always updates the first i elements of $\boldsymbol{e}_i$ simultaneously. For instance, if we are in an i-dimensional layer after computing $E_{i,i}$, we update all $E_{i,j}$ ($j = 1, \ldots, i-1$) values located in the ith row of $\boldsymbol{E}$. So, at each layer we calculate all the projection values obtained after the projection of the vector $\boldsymbol{r}_i$ to the lattice basis vectors. These values will then be used in order to update the $E_{j,j}$ ($j = 1, \ldots, i-1$) values when we are moving down in the layers. But why should one project the entire vector $\boldsymbol{r}_i$ to the lattice basis vectors, and calculate the other $E_{i,j}$ ($j = 1, \ldots, i-1$) values when they are not supposed to be used at that stage of the algorithm? The answer to this question inspires an intelligent algorithm to keep track of projection and updating the $E_{i,j}$ values.

B. Smart (Vector) Projection Method

The method proposed in Sec. III-A is far from ideal and is not, however, the most efficient way of projecting and updating the $E_{i,j}$ values. The algorithm we propose to manage the optimum projection of the received vector $\boldsymbol{r}$ is based on following criteria:

- As explained in Sec. III-A we are just interested in elements located in the lower triangular form of $\boldsymbol{E}$.
- The last row of $\boldsymbol{E}$, $\boldsymbol{e}_n$, is just calculated once (6).
- According to (14) and (15) updating every $E_{j,i}$ element ($i \leq j < n$) of $\boldsymbol{E}$ requires knowledge of both $E_{j+1,i}$ and y_{j+1} values.
- Unlike the SE enumeration strategy in [3] which follows the row-wise updating method we follow the column-wise updating strategy plus a significant refinement to it that avoids starting from the nth layer and updating all $E_{n-1,i}, E_{n-2,i}, \ldots, E_{i+1,i}$ elements located in the ith column of $\boldsymbol{E}$ before calculating the objective $E_{i,i}$ value.
- If we move to an i-dimensional layer, the first $i+1$ elements of $\boldsymbol{e}_i$ and of other $\boldsymbol{e}$ vectors above that row will be affected since we are projecting the received vector $\boldsymbol{r}$ to this new i-dimensional layer. However, the elements below $\boldsymbol{e}_i$ will remain unaffected.
- Our main target at each stage, when we are moving towards the lower-dimensional layers, is just to update the $E_{i,i}$ values. The other $E_{j,i}$ values for $j > i$ will be updated if they are needed to calculate the $E_{i,i}$ values, otherwise they will remain untouched.
- We should keep in mind every time up to which layers we have moved downwards and upwards (more explanation in Sec. IV).

According to preceding principles a novel method of projection and calculation of $E_{i,j}$ values is proposed in the next section. With the method proposed above, the elements of $\boldsymbol{E}$ would be calculated in an optimal way. In Sec. V when the simulation results are exposed we will notice how the complexity of the sphere decoder algorithm, for both lattices and finite constellations, is reduced due to optimum projection of the received vector $\boldsymbol{r}$.

Due to the evident performance gain the sphere decoder based on SE enumeration strategy possesses compared to its other predecessors, we apply herein the proposed refinement just to the SE algorithm. However, the discussed optimal projection method can be implemented to other recursive enumeration strategies including the FP enumeration method. The $E_{i,i}$ (14) and B_i (19) elements defined here are the same S_i and T_i variables explained in [4] that can be updated with the same strategy.

IV. The Proposed Algorithm

A standalone representation of the new algorithm based on the proposed smart vector projection method and the SE enumeration strategy is given. The algorithm is intended to be sufficiently detailed to allow a straightforward implementation, even without knowledge of the underlying theory. The pseudocode presented in Sec. IV-A is written for lattices. However, the modification needed for finite constellations is included in Sec. IV-B.

A. Sphere Decoder for Lattices

The input parameters of the algorithm are received vector $\boldsymbol{r}$ (5) of size n and the $n \times n$ inverse matrix $\boldsymbol{H}$ (4) of the lattice generator matrix $\boldsymbol{G}$. The output is an n-dimensional integer vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n) \in \mathbb{Z}^n$ which gives a closest lattice point $\boldsymbol{x} \in \Lambda(\boldsymbol{G}, \mathbb{Z})$ to the received vector $\boldsymbol{r}$ after a simple matrix multiplication $\boldsymbol{x} = \hat{\boldsymbol{u}}\boldsymbol{G}$. The function ***sign***(a) which is used in the algorithm returns -1 if $a \leq 0$ and 1 if $a > 0$, and the ***round*** operation rounds off everything to the nearest integer number. Ties in the ***round*** operation can be broken arbitrarily.

Algorithm Decode $(\boldsymbol{H}, \boldsymbol{r})$

1 $n = Lattice\ Dimension$ — Initialization
2 $C = \infty$
3 $k = n$
4 $d_i = k, \quad i = 1, \ldots, k-1$
5 $\lambda_{k+1} = 0$
6 $\boldsymbol{e}_k = \boldsymbol{rH}$
7 $u_k = \textbf{\textit{round}}(E_{kk})$
8 $y_k = (E_{kk} - u_k)/H_{kk}$
9 $step_k = \textbf{\textit{sign}}(y_k)$
10 $\lambda_k = y_k^2$
11 ***LOOP***
12 ***do*** { — Case(A)
13 ***if*** $(k \neq 1)$ {
14 $k = k - 1$
15 $E_{i-1,k} = E_{i,k} - y_i H_{i,k}, \quad i = d_k, d_k - 1, \ldots, k+1$
16 $u_k = \textbf{\textit{round}}(E_{kk})$
17 $y_k = (E_{kk} - u_k)/H_{kk}$
18 $step_k = \textbf{\textit{sign}}(y_k)$
19 $\lambda_k = \lambda_{k+1} + y_k^2$
20 } ***else*** {
21 $\hat{\boldsymbol{u}} = \boldsymbol{u}$
22 $C = \lambda_1$
23 }
24 } ***while*** $(\lambda_k < C)$
25 $min = k$
26 ***do*** { — Case(B)
27 ***if*** $(k = n)$
28 $return\ \hat{\boldsymbol{u}}\ and\ exit$
29 ***else*** {
30 $k = k + 1$
31 $u_k = u_k + step_k$
32 $y_k = (E_{kk} - u_k)/H_{kk}$
33 $step_k = -step_k - \textbf{\textit{sign}}(step_k)$
34 $\lambda_k = \lambda_{k+1} + y_k^2$
35 }
36 } ***while*** $(\lambda_k \geq C)$
37 $max = k$
38 $d_i = max, \quad i = max-1, max-2, \ldots, min$ — Case(C)
39 ***for*** $(i = min - 1, min - 2, \ldots, 1)$ {
40 ***if*** $(d_i < max)$
41 $d_i = max$
42 ***else***
43 $exit\ the$ ***for*** $loop$
44 }
45 $goto$ ***LOOP***

As illustrated, after the initialization part the algorithm is divided into three main subsections. We stay in Case(A) if we are moving down the layers, and the calculated Euclidean distance λ_k (18) between the received vector $\boldsymbol{r}$ and the projected vector $\boldsymbol{r}_{k-1}$ (17), is less than the Euclidean distance C between the received vector $\boldsymbol{r}$ and the closest lattice point detected so far. On the contrary, if λ_k is more than C, we move up in the hierarchy of layers. This is done in Case(B). Moreover, before quitting Case(A) and (B) each time, we store the minimum and maximum level of the layers that we have moved downwards and upwards, respectively. We put these values in variables min and max, respectively.

The algorithm to manage the projection and calculation of $E_{j,i}$ values is proposed in Case(C). The $\boldsymbol{d}$ vector is an $1 \times (n-1)$ integer vector which denotes the layer that we should start updating the $E_{j,i}$ values for $j > i$ in order to update the objective $E_{i,i}$ value. For instance, $d_i = k$ indicates that in order to update the $E_{i,i}$ value we should start the projection from kth layer, where $k > i$, and calculate all $E_{j,i}$ $(j = k-1, \ldots, i)$ elements of $\boldsymbol{E}$.

By taking out lines 4, 25, 37–44 and replacing y_k with y in lines 8–10, 17–19, 32, 34 and also replacing line 15 with

$$E_{k,i} = E_{k+1,i} - yH_{k+1,i}, \qquad i = 1, \ldots, k,$$

the proposed algorithm is changed back to the standard SE algorithm in [3].

B. Sphere Decoder for Finite Constellations

Here we present the modification needed to confine the new algorithm, proposed in Sec. IV-A, to finite constellations. The

input parameters of the algorithm are received vector $\boldsymbol{r}$ (5) of size n, the $n \times n$ inverse matrix $\boldsymbol{H}$ (4) of the channel matrix $\boldsymbol{G}$, and the minimum (U_{min}) and maximum (U_{max}) levels of the signal constellation, which are constructed from consecutive integer numbers in (2). The output is an n-dimensional integer vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n) \in \mathcal{U}^n$, which gives a closest lattice point $\boldsymbol{x} \in \Lambda(\boldsymbol{G}, \mathcal{U})$ to the received vector $\boldsymbol{r}$ after a simple matrix multiplication $\boldsymbol{x} = \hat{\boldsymbol{u}}\boldsymbol{G}$. The changes required in the algorithm of Sec. IV-A are as follows.

- Replace lines 7 and 16 with $u_k = \textbf{\textit{roundc}}(E_{kk})$
- Replace lines 31–33 with

$y_k = \infty$
$u_k = u_k + step_k$
$step_k = -step_k - \textbf{\textit{sign}}(step_k)$
if $(U_{min} \leq u_k \leq U_{max})$
  $y_k = (E_{kk} - u_k)/H_{kk}$
else $\{$
  $u_k = u_k + step_k$
  $step_k = -step_k - \textbf{\textit{sign}}(step_k)$
  if $(U_{min} \leq u_k \leq U_{max})$
    $y_k = (E_{kk} - u_k)/H_{kk}$
$\}$

The ***roundc*** operation rounds off everything to the nearest integer number inside the constellation boundary $\mathcal{U}$.

V. Simulation Results

Herein, we evaluate the effectiveness of the proposed smart vector projection technique on the sphere decoder algorithm based on SE enumeration strategy for both lattices and finite constellations. For the sake of simplicity, we use notation Algorithm(I), for our proposed algorithm in Sec. IV, and Algorithm(II), for the standard sphere decoder algorithm in [3]. Both Algorithms are implemented according to pseudocode presented in Sec. IV. The comparison results for lattices with and without considering the reduction are presented in Sec. V-A. In Sec. V-B, the two algorithms are developed and evaluated for finite constellations.

We did not use the execution time as a comparison measure to illustrate the complexity of the two algorithms since time is dependent to many different factors. Running the two algorithms under different compilation methods gives different results. Even using the same compiler but different processors changes the outcome remarkably. Apart from that, we cannot handle the programs that are running in parallel during the execution of the algorithm. Hence, to be more precise, we base our performance comparison measure on counting the number of operations that each algorithm conducts to reach the closest lattice point.

The numerical operations are divided into three different groups, floating point operations (Flops), indexing operations (Inx-Ops), and integer operations (Int-Ops). We do not use any matrix representation in our code, unlike what is presented in the pseudocode representation of the algorithm in Sec. IV, since using arrays instead of matrices are less complex and time consuming in programming languages. So, the indexing operations are based on array based programming not matrix based programming. We count any floating point addition, subtraction, multiplication, division, and comparison as floating point operations. Similarly, any integer addition, subtraction, multiplication, division, and comparison are counted towards integer operations. The indexing operations deal with array indexing. For instance, $\boldsymbol{f}(ki + j)$ addresses the $(ki + j)$th element of array $\boldsymbol{f}$ in memory unit and $ki + j$ is two indexing operations, just like the integer operations. At some points we refer to an element of an array, which has an unchanging factor several times. Specifically, if we want to refer to $\boldsymbol{f}(i + j + k)$ in a ***for*** loop and $i + j$ is the unchanging factor, as long as the value of $i + j$ is not changed we count $i + j$ as a single indexing operation no matter how many times the ***for*** loop is executed. Similarly, if two or more arrays with the same unchanging $i + j$ factor are addressed, just a single indexing operation is counted. On the other hand, we do not count the ***for*** loop counters towards any kinds of numerical operations since the concept of the counters are different in programming languages and are not considered as numerical operations. The ***round*** operation is counted as a single floating point operation and ***roundc*** in Sec. V-B is counted as one and two floating point operations when 2-PAM and 4-PAM constellations are investigated, respectively.

To compare the complexity of the two algorithms, we generate M random generator matrices $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_M$, and for each matrix $\boldsymbol{G}_j$ we generate N random received vectors $\boldsymbol{r}_{j,1}, \ldots, \boldsymbol{r}_{j,N}$. The same vectors are decoded using both algorithms and the operations are counted.

Depending on various kinds of applications, three different types of averaging schemes for comparing the complexity between the two algorithms are

$$Av_1 = \frac{1}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} \frac{Ops_{II}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}{Ops_I(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}, \tag{21}$$

$$Av_2 = \frac{1}{M} \sum_{j=1}^{M} \frac{\sum_{i=1}^{N} Ops_{II}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}{\sum_{i=1}^{N} Ops_I(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}, \tag{22}$$

$$Av_3 = \frac{\sum_{j=1}^{M} \sum_{i=1}^{N} Ops_{II}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}{\sum_{j=1}^{M} \sum_{i=1}^{N} Ops_I(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}, \tag{23}$$

where $Ops_I(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)$ and $Ops_{II}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)$ denote the total number of operations that Algorithm(I) and (II) carry out respectively to decode the received vector $\boldsymbol{r}_{j,i}$ in the lattice generated with generator matrix $\boldsymbol{G}_j$. In our case, Ops_I and Ops_{II} can be any kinds of numerical operations discussed above.

A. Lattices

We generate the lattice generator matrices with random numbers, drawn from i.i.d zero mean, unit variance Gaussian distributions. The random input vectors are generated uniformly inside a Voronoi region according to [27].

Our simulation results are based on averaging over $M = 50$ different generator matrices. However, the numbers of input vectors is related to the dimension of the lattices. The higher the dimensions are, the fewer input vectors are examined, to the extent that we assure the plotted curves are reasonably smooth.

TABLE I
AVERAGE NUMBER OF NUMERICAL OPERATIONS ACCORDING TO EQUATION (24) WITHOUT ANY REDUCTION

Dimension	Algorithm(I)			Algorithm(II)		
	Flops	Inx-Ops	Int-Ops	Flops	Inx-Ops	Int-Ops
5	3.2e+2	2.0e+2	1.6e+2	3.4e+2	2.4e+2	1.5e+2
10	6.0e+3	3.6e+3	2.9e+3	8.1e+3	6.1e+3	2.7e+3
20	2.1e+5	1.4e+5	1.0e+5	3.9e+5	3.2e+5	0.9e+5
30	1.5e+7	1.0e+7	0.7e+7	3.7e+7	3.3e+7	0.6e+7
40	6.0e+8	3.8e+8	2.7e+8	18.2e+8	16.4e+8	2.4e+8
50	4.1e+10	2.6e+10	1.8e+10	15.3e+10	14.1e+10	1.6e+10
60	8.4e+11	5.4e+11	3.8e+11	34.9e+11	32.2e+11	3.3e+11

Tab. I gives the average numbers of numerical operations,

$$Av = \frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N} Ops(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j), \tag{24}$$

for both algorithms and all three types of operations. It illustrates that most of operations in the two algorithms are based on Flops and Inx-Ops, especially as the dimension increases. For instance, at dimension 60 in the standard Algorithm(II) the Flops and Inx-Ops are roughly 10 times more than the Int-Ops. As a result, it can be concluded that Flops and Inx-Ops are dominant factors in the complexity of the algorithms.

The lattices have also passed thorough a preprocessing stage and the complexity is compared once the so-called LLL reduced basis are obtained. Figs. 3–5 are plotted for different types of numerical operations based on the Av_1 averaging scheme in (21), without taking into account the operations needed for the reduction. Figs. 3–4 show that the gain with Algorithm(I) increases linearly with the dimension, for both Flops and Inx-Ops. The drawback is a somewhat larger number of Int-Ops, as shown in Fig. 5, but the penalty converges to a mere 15% increase at high dimensions, which according to Tab. I has a quite small effect on overall complexity. The usage of reduction does not change the ratios substantially, although the absolute numbers decrease.

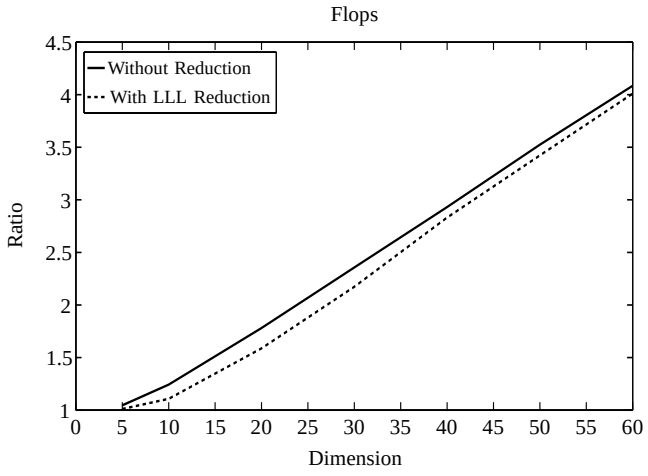

Fig. 3. Comparison of average floating point operations between Algorithm(I) and Algorithm(II) based on the Av_1 averaging scheme in (21).

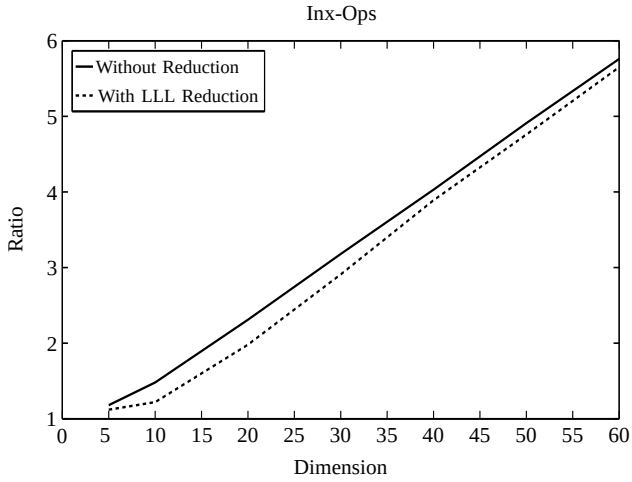

Fig. 4. Comparison of average indexing operations between Algorithm(I) and Algorithm(II) based on the Av_1 averaging scheme in (21).

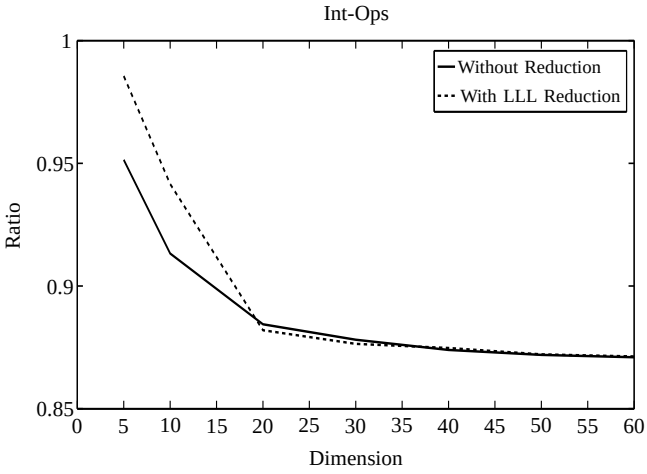

Fig. 5. Comparison of average integer operations between Algorithm(I) and Algorithm(II) based on the Av_1 averaging scheme in (21).

Plotting the figures according to other proposed averaging schemes, one would notice that Av_2 (22) curves are exactly the same as the plotted Av_1 curves, while in case of Av_3 (23) we might face some minor fluctuations, see Tab. I, due to domination of a few ill-conditioned lattice generator matrices.

B. Finite Constellations

In communications theory, one of the most important applications where the name of the sphere decoder algorithm arises is ML detection for MIMO channels. Assuming perfect channel estimation over fading conditions, at each stage we have a channel matrix $\boldsymbol{G}$ which changes constantly over the time axis, depending on the speed of the channel variations. Considering that the momentum of these variations are almost constant during the data transmission, the best method to compare the performance of the two proposed ML decoder algorithms is the second averaging scheme Av_2 in (22). Hence, according to Sec. IV the Algorithms(I) and (II) are constructed for finite constellations, and the complexity of them is compared for 2-PAM and 4-PAM finite constellations

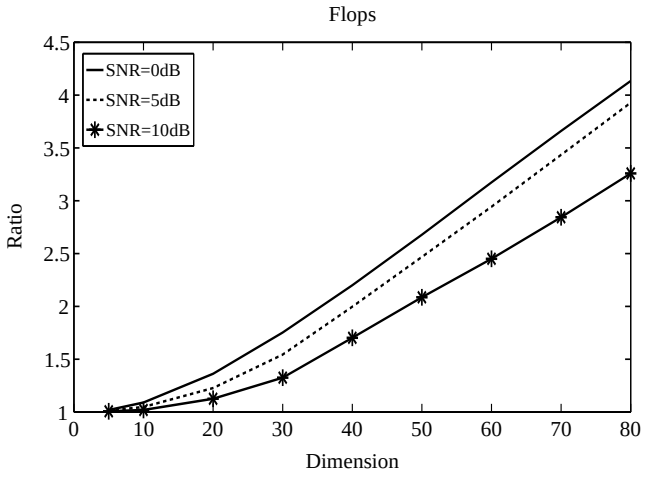

Fig. 6. Comparison of average floating point operations between Algorithm(I) and Algorithm(II) for 2-PAM constellation for different SNRs based on the Av_2 averaging scheme in (22).

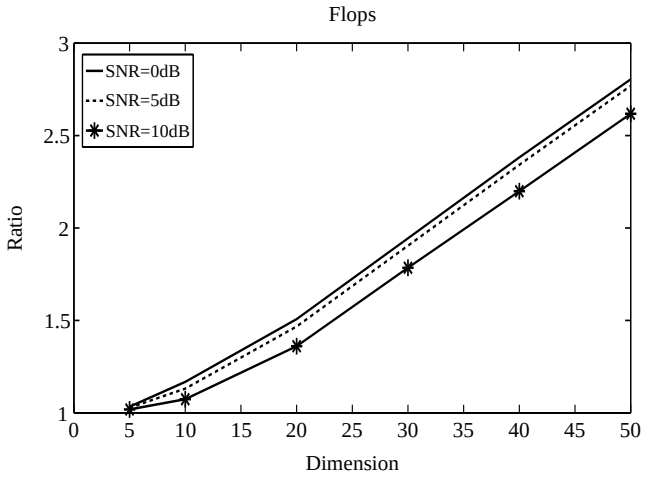

Fig. 7. Comparison of average floating point operations between Algorithm(I) and Algorithm(II) for 4-PAM constellation for different SNRs based on the Av_2 averaging scheme in (22).

without considering any reduction.

The system model in (3) for an L-PAM constellation is considered, where the average symbol energy of the constellation, E_s, is calculated from the signal set $\{-\frac{L-1}{2}, -\frac{L-1}{2}+1, \ldots, \frac{L-1}{2}\}$ and the SNR is defined as E_b/N_0, where $E_b = E_s/\log_2 L$ is the average energy per bit and $N_0/2$ is the double-sided noise spectral density.

The presented results here are just based on floating point operations, averaged over 50 different zero mean, unit variance Gaussian channel matrices. Fig. 7 illustrates that in 4-PAM constellation as the SNR increases, the curves diverges less compared to the 2-PAM constellation in Fig. 6. In all cases, the Flops increases linearly with the dimensions, and the gain is higher at low SNRs. The results presented for higher dimensions are applicable to coded transmission [28].

VI. Conclusion

In this paper we have investigated of sate-of-the-art sphere decoders and removed most of the numerical operations without compromising the performance. The algorithm in Sec. IV-B performs ML detection in finite lattice subsets and is hence suitable for MIMO detection, while the variant in Sec. IV-A performs closest-point search in (infinite) lattices. We believe that the new algorithm is the fastest algorithms known for both purposes.

References

[1] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*. New York: Springer-Verlag, 1988.

[2] ——, "Fast quantizing and decoding and algorithms for lattice quantizers and codes," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 227–232, Mar. 1982.

[3] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[4] E. Viterbo and J. J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1639–1642, July 1999.

[5] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," *SIGSAM Bulletin*, vol. 15, no. 1, pp. 37–44, Feb. 1981.

[6] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of Computation*, vol. 44, no. 170, pp. 463–471, Apr. 1985.

[7] E. Viterbo and E. Biglieri, "A universal lattice decoder," *GRETSI 14-ème Colloque, Juan-les-Pins, France*, Sept. 1993.

[8] C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, no. 2, pp. 181–199, 1994.

[9] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[10] W. H. Mow, "Universal lattice decoding: principle and recent advances," *Wireless Communications and Mobile Computing*, vol. 3, no. 5, pp. 553–569, 2003.

[11] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Transactions on Information Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[12] B. Shim and I. Kang, "Sphere decoding with a probabilistic tree pruning," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4867–4878, Oct. 2008.

[13] R. Gowaikar and B. Hassibi, "Statistical pruning for near-maximum likelihood decoding," *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 2661–2675, June 2007.

[14] W. Zhao and G. B. Giannakis, "Sphere decoding algorithms with improved radius search," *IEEE Transactions on Communications*, vol. 53, no. 7, pp. 1104–1109, July 2005.

[15] ——, "Reduced complexity closest point decoding algorithms for random lattices," *IEEE Transactions on Wireless Communications*, vol. 5, no. 1, pp. 101–111, Jan. 2006.

[16] O. Damen, A. Chkeif, and J. C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Communications Letters*, vol. 4, no. 5, pp. 161–163, May 2000.

[17] J. J. Boutros, N. Gresset, L. Brunel, and M. Fossorier, "Soft-input soft-output lattice sphere decoder for linear channels," in *Proceedings IEEE GLOBECOM*, vol. 3, San Francisco, CA, Dec. 2003, pp. 1583–1587.

[18] W. K. Ma, B. N. Vo, T. N. Davidson, and P. C. Ching, "Blind ML detection of orthogonal space-time block codes: efficient high-performance implementations," *IEEE Transactions on Signal Processing*, vol. 54, no. 2, pp. 738–751, Feb. 2006.

[19] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, July 2005.

[20] E. Agrell and T. Eriksson, "Optimization of lattices for quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 5, pp. 1814–1828, Sept. 1998.

[21] B. M. Hochwald, C. B. Peel, and A. L. Swindlehurst, "A vector-perturbation technique for near-capacity multiantenna multiuser communication-part II: perturbation," *IEEE Transactions on Communications*, vol. 53, no. 3, pp. 537–544, Mar. 2005.

[22] L. Brunel and J. J. Boutros, “Lattice decoding for joint detection in direct-sequence CDMA systems,” *IEEE Transactions on Information Theory*, vol. 49, no. 4, pp. 1030–1037, Apr. 2003.
[23] L. Lampe, R. Schober, V. Pauli, and C. Windpassinger, “Multiple-symbol differential sphere decoding,” *IEEE Transactions on Communications*, vol. 53, no. 12, pp. 1981–1985, Dec. 2005.
[24] M. S. Yee, “Max-log-map sphere decoder,” in *Proceedings IEEE ICASSP*, vol. 3, Piscataway, NJ, Mar. 2005, pp. 1013–1016.
[25] L. Babai, “On Lovász’ lattice reduction and the nearest lattice point problem,” *Combinatorica*, vol. 6, no. 1, pp. 1–13, 1986.
[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.
[27] J. H. Conway and N. J. A. Sloane, “On the Voronoi regions of certain lattices,” *SIAM Journal on Algebraic and Discrete Methods*, vol. 5, no. 3, pp. 294–305, Sept. 1984.
[28] B. Hassibi and B. M. Hochwald, “High-rate codes that are linear in space and time,” *IEEE Transactions on Information Theory*, vol. 48, no. 7, pp. 1804–1824, Jul. 2002.